AMATH 482 Homework 5

Wenxuan Liu

March 17, 2021

Abstract

We have two videos, one records a car race, and the other records a skier. We want to isolate the moving objects from backgrounds for both videos. We are going to use the Dynamic Mode Decomposition (DMD) to analyze the video. We will treat the video as a discretely sampled frames, and we use low-rank DMD to separate the constant from dynamic part of the video frames. Then we can subtract the background from the original frames and get the moving fore-ground. We are also going to plot the eigenvalues (Omega) of the Koopman operator to see how much background need to be separated.

1 Introduction and Overview

We are given two videos, first one is a car race video where there is a background of race track, and there are cars going through. The second one is a skier drop from mountain top, and a mountain view background. The videos are both complex dynamic system moving in time and space. The videos also have lots of data (pixel) in them, which is a lot of information to process. We are going to get the background and fore ground separated using Dynamic Mode Decomposition (DMD). DMD is a technique that allow us to simplify the dynamic system by taking the major variation from it and transform the movement into oscillations, decay, and/or growth. Most importantly, the eigenvalues of the Koopman operator tells us what type of movement each modes are responsible for. We can spot the modes that never move, which is the key to extract the background out of the video. After getting the background, we can subtract it from the original video frames to get a foreground.

2 Theoretical Background

2.1 Koopman Operator

From text book (or lecture 26) [1], we learned Koopman Operator. A Koopman Operator A is a linear, time independent operator such that

$$X_{j+1} = AX_j$$

Where j is the j^{th} snapshot timestamp. The matrix A is a linear transformation that describes the transformation from one timestamp to the next one. This linear transformation is global, which means A is a constant matrix over space or time. The linear transformation can be applied to nonlinear dynamic systems.

2.2 Dynamic Mode Decomposition

From text book (or lecture 26) [1], we learned Dynamic Mode Decomposition (DMD). The DMD method approximates the modes of Koopman Operator that can describe the change of the data samples over time and space, and even predict the data in the future. DMD depends exclusively on the data collected at each timestamp, so we don't have to know any equation that describe the dynamic system in order to approximate the system. To construct DMD modes that approximate the Koopman Operator of a dynamic system, we follow the following steps:

Firstly, we construct a matrix, where X_j is the snapshot of data at time j:

$$X_1^{M-1} = [X_1 X_2 \dots X_{M-1}] \tag{1}$$

By definition of Koopman operator, we can rewrite the matrix as:

$$X_1^{M-1} = [X_1 A X_1 A^2 X_1 \dots A^{M-2} X_1] \tag{2}$$

Now we have a relationship between initial state and all $M-1$ snapshots. We rewrite it again to get the following relationship with X_2^M, which is "next state" of all snapshots in X_1^{M-1}:

$$X_2^M = A X_1^{M-1} + r e_{M-1}^T \tag{3}$$

Where e_{M-1} is zero vector with 1 at position $M-1$. We add this error term to account for that x_M is not included in X_1^{M-1}, or Krylov subspace. The behavior of Koopman operator is encoded in its eigenvalues and eigenvectors. Thus, we are going to use the above relationship to derive a matrix $\tilde{S}$ that has same eigenvalues as A, which also gives us eigenvectors later on. To find the $\tilde{S}$, we first take the SVD of X_1^{M-1} and substitute into above equation to get:

$$\begin{aligned}
X_2^M &= A X_1^{M-1} + r e_{M-1}^T && \text{[Original Relationship]}\\
X_2^M &= A U \Sigma V^* + r e_{M-1}^T && \text{[Substitute]}\\
U^* X_2^M &= U^* A U \Sigma V^* && \text{[Multiply on both sides]}\\
U^* X_2^M V \Sigma^{-1} &= U^* A U && \text{[Multiply on both sides]}\\
U^* X_2^M V \Sigma^{-1} &= \tilde{S} && \text{[We call LHS } \tilde{S}\text{]}
\end{aligned}$$

Note that we require the error r to be zero because we required X_2^M to be linear combination of U. This above relationship between $\tilde{S}$ and A shows that A is multiplied by U and its inverse on both sides. This is a sign of A is similar to $\tilde{S}$ in a way that they have same eigenvalues, and more importantly, if y is an eigenvector of $\tilde{S}$, Uy is eigenvector of A. Thus, we also get k^{th} eigenvector of A:

$$\psi_k = U y_k \tag{4}$$

We can just expand in our eigenbasis to get

$$x_{DMD}(t) = \sum_{k=1}^{K} b_k \psi_k e^{\omega_k t} = \Psi diag(e^{\omega_k t}) \boldsymbol{b} \tag{5}$$

Where K is rank of X_1^{M-1}, b_k is the initial amplitude of each mode, and Ψ contains all eigenvectors of A. we write the time dynamics as exponential functions, meaning that $\omega_k = ln(\mu_k)/\Delta t$. In the real world, time is continuous and so we can interpolate between time jumps with the above formula (guess what happens between each timestamp). Lastly, to get $\boldsymbol{b}$, we need the initial state of dynamic system:

$$\begin{aligned}
x_1 &= \Psi \boldsymbol{b} && [\boldsymbol{b} \text{ is eigenvector of initial state}]\\
\boldsymbol{b} &= \Psi^\dagger x_1 && [\text{Take pseudo inverse of } \Psi]
\end{aligned}$$

3 Algorithm Implementation and Development

3.1 Monte Carlo Case

1. First step is to load the video into MatLab and process the video into a 2-D matrix. We use videoReader to read in all frames of the video. Then we loop through all frames to turn each frame to gray and reshape to row vector. We get a matrix where each row is a frame, and we treat this matrix as our samples. The videoReader also tell us the FrameRate and Duration, which we use to define variables t and dt.

2. Next, we are going to use DMD to get the low rank DMD solution of X, our video matrix. To use DMD, we define $X_1,\ X_2$ to be the $(1 : end-1)^{th},\ (2 : end)^{th}$ columns of X. We get SVD decomposition of X_1. While the rank of SVD modes is equal to the number of frames, we actually only need the first few modes to reconstruct the background. To get the exact number of rank we need, we look at the singular values after ploting them.

3. After picking the rank, we truncate the SVD to calculate the $\tilde{S}$ that is similar to the Koopman operator. Then we can use the MatLab function $eig()$ to get the eigenvalues and eigenvectors of similar matrix $\tilde{S}$. Near zero eigenvalues of the Koopman operator indicate a non-changing component of the dynamic system. Thus, we inspect the eigenvalues to keep only those modes corresponding to the near zero eigenvalues.

4. The last element we need is initial condition of all dmd modes. We calculate it by $\boldsymbol{b} = Phi\ X1(:,1)$, and now we get all quantities on the RHS of the equation:

$$x_{DMD}(t) = \sum_{k=1}^{K} b_k \psi_k e^{\omega_k t} = \Psi diag(e^{\omega_k t})\boldsymbol{b} \tag{6}$$

   We loop through each timestamp t and reconstruct the Low-Rank-DMD solution at each timestamp just using those modes with corresponding eigenvalues near 0. We store the result in a matrix.

5. We are done with getting background matrix, and for completeness, we want to also get the foreground. We subtract the background matrix from the original data we collected, and this result should capture the foreground moving objects. We reshape the back/fore-ground matrices to the original shape $height \times width \times frames$ and put residual negative the residual matrix R and subtract it from foreground (or sparse matrix). Lastly, project them onto $(0, 1)$ using $mat2gray()$ and write the frames into files.

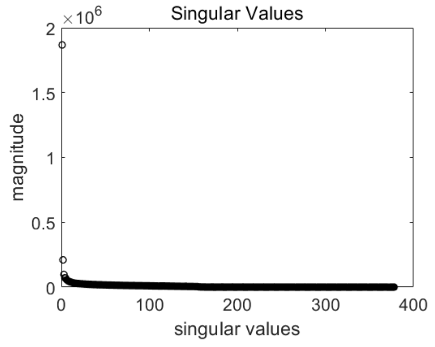
3.2 Ski Drop Case

Repeat everything we did to Monte Carlo case. Only two constants are changed: the number of ranks we choose to build similar matrix $\tilde{S}$, and the threshold of picking near zero eigenvalues.

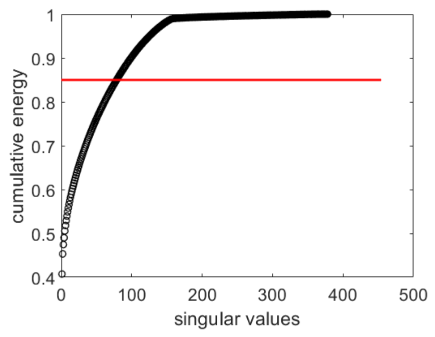
4 Computational Results

4.1 Monte Carlo Case

We take the SVD of X_1, and we inspect the eigenvalues by plotting all eigenvalues and the cumulative sum:



(a) Singular values of X1



(b) Cumulative sum of singular values

Figure 1: PCA Analysis

Looking at the singular values, we want to include enough information about the back ground, so we pick first about 75 SVD modes to create the similar matrix $\tilde{S}$.

We get enough background information in $\tilde{S}$, so we can compute the eigenvalues of $\tilde{S}$, which is shown below:

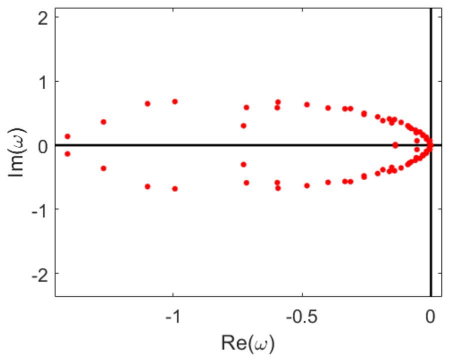


Figure 2: All eigenvalues of $\tilde{S}$

The first 10 smallest eigenvalues are:

$$0.0021, 0.6854, 0.6854, 2.2051, 2.2051, 3.1192, 3.1192, 4.0943, 4.0943, 5.2315$$

We can see that there is only one eigenvalue close to zero, and the others are far away. Thus, we will use the mode corresponding to this single eigenvalue to reconstruct the DMD solution.
The result of the reconstruction:

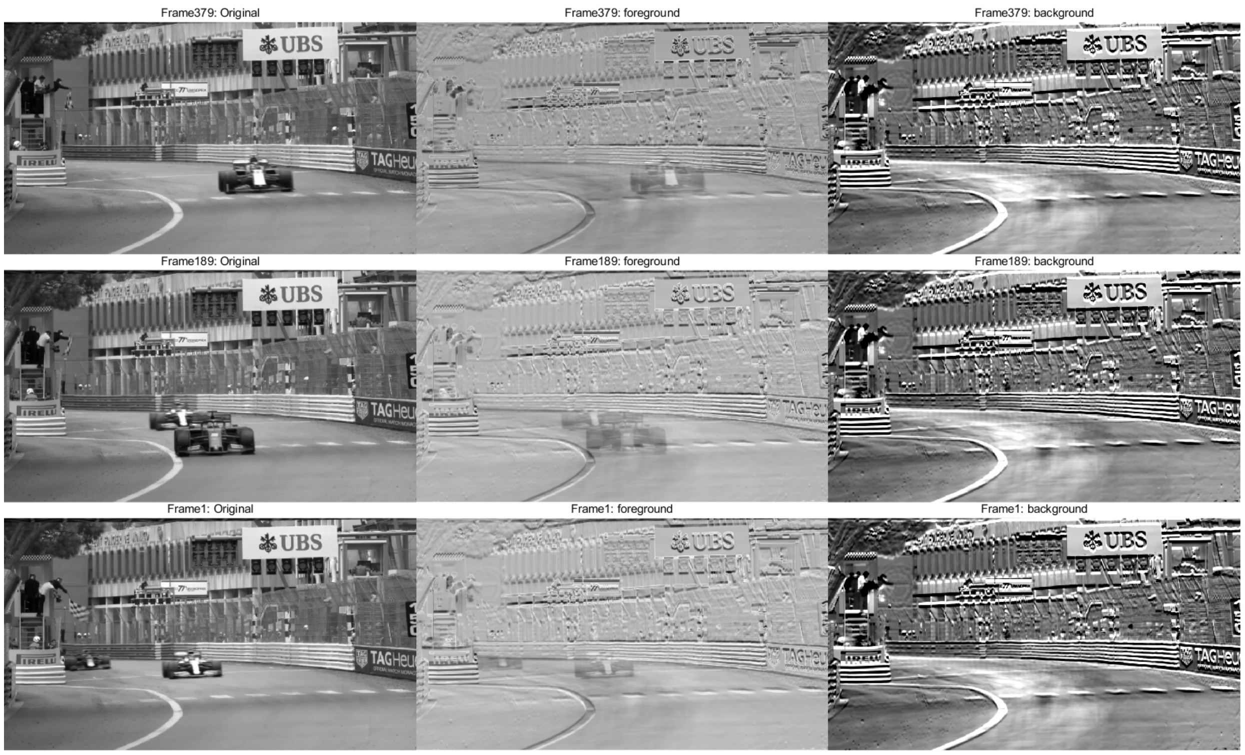
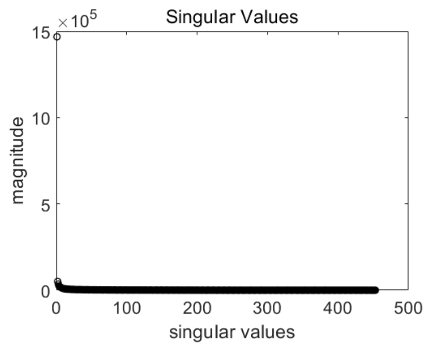


Figure 3: Compare three frames over back/fore/original

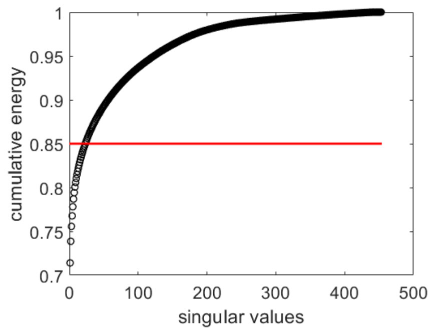
From the picture, we can see a clear separation between background and foreground. In background video, the racing track is completely still without racing car or flying flag. In foreground video, the car and flag are very detail, and the background that is not moving is mostly wiped out.

4.2 Ski Drop Case

We take the SVD of X_1, and we inspect the eigenvalues by plotting all eigenvalues and the cumulative sum:



(a) Singular values of X1



(b) Cumulative sum of singular values

Figure 4: PCA Analysis

Looking at the singular values, we want to include enough information about the back ground, so we pick first about 20 SVD modes to create the similar matrix $\tilde{S}$. We get enough background information in $\tilde{S}$, so we can compute the eigenvalues of $\tilde{S}$, which is shown below:

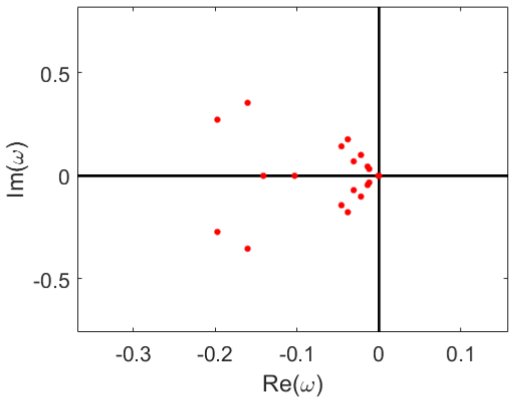


Figure 5: All eigenvalues of $\tilde{S}$

The first 10 smallest eigenvalues are:

$$0.0004, 0.5330, 0.5330, 1.2622, 1.2622, 2.0986, 2.0986, 2.8117, 2.8117, 3.4768$$

We can see that there is only one eigenvalue close to zero, and the others are far away. Thus, we will use the mode corresponding to this single eigenvalue to reconstruct the DMD solution.
The result of the reconstruction:

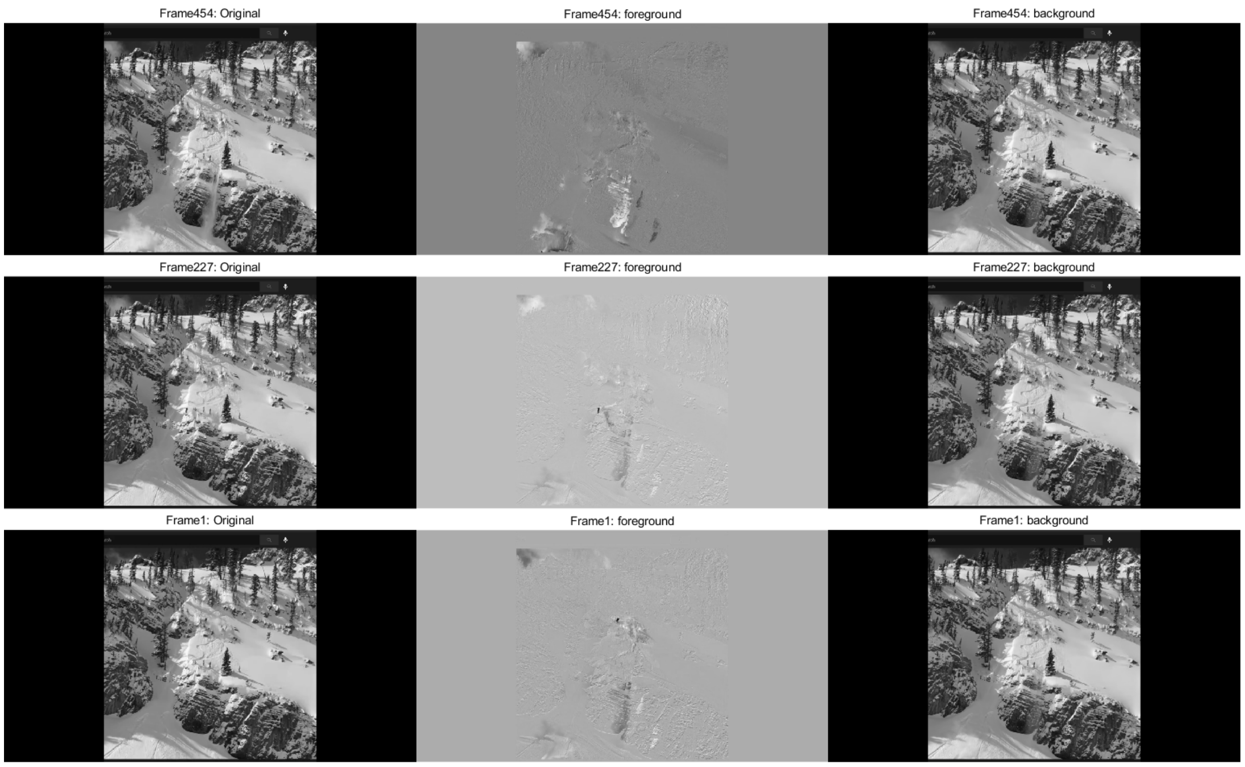


Figure 6: Compare three frames over back/fore/original

From the picture, we can see a clear separation between background and foreground. In background video, the mountain is completely still without the man and falling snow. In foreground video, the man and falling snow are very clear and detailed, and the background that is not moving is mostly wiped out.

5 Summary and Conclusions

To this point, we have well separated the foreground and background in both Monte Carlo and Ski Drop videos. The background is free of moving objects, which are all captured by the foreground matrix. The background separation is very clear and satisfying; however, the foreground is not as clear as background. We can still see outline of the background in foreground because the camera is not stable, so the small global movement in background is captured as foreground.

References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

Appendix A MATLAB Functions

- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix A, such that A = U*S*V'.
- `D = diag(v)` returns a square diagonal matrix with the elements of vector v on the main diagonal.
- `imshow(I)` displays the grayscale image I in a figure. imshow uses the default display range for the image data type and optimizes figure, axes, and image object properties for image display.
- `C = bsxfun(fun,A,B)` applies the element-wise binary operation specified by the function handle fun to arrays A and B.
- `B = reshape(A,sz)`reshapes A using the size vector, sz, to define size(B). For example, reshape(A,[2,3]) reshapes A into a 2-by-3 matrix. sz must contain at least 2 elements, and prod(sz) must be the same as numel(A).
- `I2 = im2double(I)`converts the image I to double precision. I can be a grayscale intensity image, a truecolor image, or a binary image. im2double rescales the output from integer data types to the range [0, 1].
- `k = find(X)`returns a vector containing the linear indices of each nonzero element in array X.
  If X is a vector, then find returns a vector with the same orientation as X.
  If X is a multidimensional array, then find returns a column vector of the linear indices of the result.
- `plot3(X,Y,Z)`plots coordinates in 3-D space.
  To plot a set of coordinates connected by line segments, specify X, Y, and Z as vectors of the same length.
  To plot multiple sets of coordinates on the same set of axes, specify at least one of X, Y, or Z as a matrix and the others as vectors.
- `tree = fitctree(Tbl,ResponseVarName)`returns a fitted binary classification decision tree based on the input variables (also known as predictors, features, or attributes) contained in the table Tbl and output (response or labels) contained in Tbl.ResponseVarName. The returned binary tree splits branching nodes based on the values of a column of Tbl.
- `[labelIdx,score] = predict(categoryClassifier,I)`returns the predicted label index and score for the input image.
- `Mdl = fitcecoc(Tbl,ResponseVarName)`returns a full, trained, multiclass, error-correcting output codes (ECOC) model using the predictors in table Tbl and the class labels in Tbl.ResponseVarName. fitcecoc uses $K(K-1)/2$ binary support vector machine (SVM) models using the one-versus-one coding design, where K is the number of unique class labels (levels). Mdl is a ClassificationECOC model.
- `S = sum(A)` returns the sum of the elements of A along the first array dimension whose size does not equal 1.
  If A is a vector, then sum(A) returns the sum of the elements.
  If A is a matrix, then sum(A) returns a row vector containing the sum of each column.
  If A is a multidimensional array, then sum(A) operates along the first array dimension whose size does not equal 1, treating the elements as vectors. This dimension becomes 1 while the sizes of all other dimensions remain the same.
- `B = categorical(A)` creates a categorical array from the array A. The categories of B are the sorted unique values from A.
- `B = reordercats(A)` reorders the categories in the categorical array, A, to be in alphanumeric order.
- `bar(y)` creates a bar graph with one bar for each element in y. If y is an m-by-n matrix, then bar creates m groups of n bars.
- `e = eig(A)` returns a column vector containing the eigenvalues of square matrix A.

- `n = norm(v,p)` returns the generalized vector p-norm.
- `B = sort(A)` sorts the elements of A in ascending order.
  If A is a vector, then sort(A) sorts the vector elements.
  If A is a matrix, then sort(A) treats the columns of A as vectors and sorts each column.
  If A is a multidimensional array, then sort(A) operates along the first array dimension whose size does not equal 1, treating the elements as vectors.
- `histogram(X)`creates a histogram plot of X. The histogram function uses an automatic binning algorithm that returns bins with a uniform width, chosen to cover the range of elements in X and reveal the underlying shape of the distribution. histogram displays the bins as rectangles such that the height of each rectangle indicates the number of elements in the bin.
- `Mdl = fitclinear(X,Y)`returns a trained linear classification model object Mdl that contains the results of fitting a binary support vector machine to the predictors X and class labels Y.

Appendix B MATLAB Code

MATLAB code that produce the outcomes discussed in this paper.

```
%% task1
% Clean workspace
clear all; close all; clc
v = VideoReader('monte_carlo_low.mp4'); %ski_drop_low monte_carlo_low

dt = 1 / v.FrameRate;
t = 0:dt:v.Duration;
frames = read(v);
sizeFrames = size(frames);
twoDframes = zeros(sizeFrames(4), sizeFrames(1) * sizeFrames(2));
for j = 1:v.NumFrames
    image = rgb2gray(frames(:, :, :, j));
    image = reshape(image, [1, sizeFrames(1) * sizeFrames(2)]);
    twoDframes(j, :) = image;
end
X = twoDframes';
X1 = X(:,1:end-1);
X2 = X(:,2:end);

[U, Sigma, V] = svd(X1,'econ');

rank = 20; %20 for skier video, 75 for car video.
U_R = U(:, 1:rank);
Sigma_R = Sigma(1:rank, 1:rank);
V_R = V(:, 1:rank);

S = U_R'*X2*V_R*diag(1./diag(Sigma_R));
[eV, D] = eig(S); % compute eigenvalues + eigenvectors
mu = diag(D); % extract eigenvalues
omega = log(mu)/dt;
Phi = U_R*eV;

zeroOmegas = find(abs(omega) < 0.01);
y0 = Phi\X1(:,1); % pseudoinverse to get initial conditions

u_modes = zeros(length(zeroOmegas),length(t));
for iter = 1:length(t)
    u_modes(:,iter) = y0(zeroOmegas, 1).*exp(omega(zeroOmegas, 1)*t(iter));
```

```
end
X_low_rank_dmd = Phi(:, zeroOmegas)*u_modes;

X_sparse = X - abs(X_low_rank_dmd);
%R = X_sparse .* (X_sparse < 0);
%X_low_rank_dmd = R + abs(X_low_rank_dmd);
%X_sparse = X_sparse - R;

%%
uToVideo1 = uint8(reshape(X_low_rank_dmd, [sizeFrames(1), sizeFrames(2), sizeFrames(4)]));
vw = VideoWriter('carRace_back');
open(vw)
for j = 1:v.NumFrames
    writeVideo(vw,mat2gray(uToVideo1(:,:,j)));
end
close(vw)

%X_sparse = filter2(fspecial('sobel'), X_sparse .* -1);
uToVideo2 = reshape(X_sparse, [sizeFrames(1), sizeFrames(2), sizeFrames(4)]);
vw = VideoWriter('carRace_obj');
open(vw)
for j = 1:v.NumFrames
    writeVideo(vw,mat2gray(uToVideo2(:,:,j)));
end
close(vw)

%%
slices = [1 floor(v.NumFrames/2) v.NumFrames];
uToVideo1 = uint8(reshape(X_low_rank_dmd, [sizeFrames(1), sizeFrames(2), sizeFrames(4)]));
uToVideo2 = reshape(X_sparse, [sizeFrames(1), sizeFrames(2), sizeFrames(4)]);
uToVideo3 = reshape(X, [sizeFrames(1), sizeFrames(2), sizeFrames(4)]);

figure()

for j = 1:3
    subplot(3 ,3 , 3*(j-1)+1)
    imshow(mat2gray(uToVideo1(:,:,slices(j))));
    title(sprintf('Frame%s: background',string(slices(j))))
    subplot(3 ,3 , 3*(j-1)+2)
    imshow(mat2gray(uToVideo2(:,:,slices(j))));
    title(sprintf('Frame%s: foreground',string(slices(j))))
    subplot(3 ,3 , 3*(j-1)+3)
    imshow(mat2gray(uToVideo3(:,:,slices(j))));
    title(sprintf('Frame%s: Original',string(slices(j))))
end
 ha=get(gcf,'children');
 set(ha(1),'position',[0 .66 .33 .33])%left bottom width height
 set(ha(2),'position',[.33 .66 .33 .33])
 set(ha(3),'position',[.66 .66 .33 .33])
 set(ha(4),'position',[0 .33 .33 .33])
 set(ha(5),'position',[.33 .33 .33 .33])
 set(ha(6),'position',[.66 .33 .33 .33])
 set(ha(7),'position',[0 0 .33 .33])
 set(ha(8),'position',[.33 0 .33 .33])
 set(ha(9),'position',[.66 0 .33 .33])
%% Plotting Eigenvalues (omega)

% make axis lines
line = -15:15;
```

```
plot(zeros(length(line),1),line,'k','Linewidth',2) % imaginary axis
hold on
plot(line,zeros(length(line),1),'k','Linewidth',2) % real axis
plot(real(omega)*dt,imag(omega)*dt,'r.','Markersize',15)
xlabel('Re(\omega)')
ylabel('Im(\omega)')
set(gca,'FontSize',16,'Xlim',[-1.5 0.5],'Ylim',[-3 3])

%%
plot(diag(Sigma),'ko','Linewidth',1)
set(gca,'Fontsize',16)
title('Singular Values')
xlabel('singular values')
ylabel('magnitude')

%%
plot(cumsum(diag(Sigma)./sum(diag(Sigma))),'ko','Linewidth',1), hold on
plot([0 454],[0.85, 0.85], '-r', 'linewidth', 2)
set(gca,'Fontsize',16)
xlabel('singular values')
ylabel('cumulative energy')
%%
tmp = Phi';
waterfall([1:sizeFrames(1) * sizeFrames(2)],1:10,abs(tmp(1:10, :))), colormap([0 0 0])
xlabel('x')
ylabel('modes')
zlabel('|u|')
title('DMD Modes')
set(gca,'FontSize',16)
```